Int. J. Appl. Math. Comput. Sci., 2017, Vol. 27, No. 2, 385–399
DOI: 10.1515/amcs-2017-0027

A HYBRID SCHEDULER FOR MANY TASK COMPUTING IN BIG DATA SYSTEMS

LAURA VASILIU [a], FLORIN POP [a,c,*], CATALIN NEGRU [a], MARIANA MOCANU [a], VALENTIN CRISTEA [a], JOANNA KOLODZIEJ [b]

[a]Computer Science Department, Faculty of Automatic Control and Computers
University Politehnica of Bucharest, 313, Splaiul Independentei, 060042 Bucharest, Romania
e-mail: laura.vasiliu@hpc.pub.ro, {florin.pop,catalin.negru}@cs.pub.ro,
{mariana.mocanu,valentin.cristea}@cs.pub.ro

[b]Institute of Computer Science
Cracow University of Technology, ul. Warszawska 24, 31-155 Cracow, Poland
e-mail: jokolodziej@pk.edu.pl

[c]National Institute for Research and Development in Informatics (ICI)
8–10, Mareşal Averescu, 011455 Bucharest, Romania
e-mail: florin.pop@ici.ro

With the rapid evolution of the distributed computing world in the last few years, the amount of data created and processed has fast increased to petabytes or even exabytes scale. Such huge data sets need data-intensive computing applications and impose performance requirements to the infrastructures that support them, such as high scalability, storage, fault tolerance but also efficient scheduling algorithms. This paper focuses on providing a hybrid scheduling algorithm for many task computing that addresses big data environments with few penalties, taking into consideration the deadlines and satisfying a data dependent task model. The hybrid solution consists of several heuristics and algorithms (min-min, min-max and earliest deadline first) combined in order to provide a scheduling algorithm that matches our problem. The experimental results are conducted by simulation and prove that the proposed hybrid algorithm behaves very well in terms of meeting deadlines.

Keywords: many task computing, scheduling heuristics, QoS, big data systems, simulation.

1. Introduction

Task scheduling and resource allocation represent a fundamental research area in distributed systems. Optimization of scheduling process and data distribution have impact on minimization of operational costs of data processing and transmission (Chmaj *et al.*, 2012; Benziani *et al.*, 2014; Różycki *et al.*, 2016). Also, by allocating a limited set of resources for a specific number of jobs, to achieve cost reduction, minimization of their execution failure probability and total completion time is mandatory (Janiak *et al.*, 2013; Gąsior and Seredyński, 2015). Furthermore, there are efforts to reduce network traffic (Cabrera *et al.*, 2016) in order to minimize the amount of energy consumed by a telecommunication infrastructure (Jaskóła *et al.*, 2016). Extensive experimental results showed high efficiency of meta-heuristics in solving various computation models (like many read/write tasks for game-based applications), when there is additional cost for secure task scheduling (Kołodziej and Xhafa, 2011). Energy-aware data-intensive computing for dynamic systems (Niewiadomska-Szynkiewicz *et al.*, 2014) depends on multi-control and security facilities (Karpowicz *et al.*, 2015) with an important impact on publish/subscribe services and stream processing (Dimitriou *et al.*, 2013; Bourdena *et al.*, 2014; Esposito *et al.*, 2015; Mavromoustakis *et al.*, 2015). These services face with reliability requests

*Corresponding author

for data harvesting (Esposito *et al.*, 2013; 2014; He *et al.*, 2016) or stream processing considering specific cloud platforms.

In the world of big data, a key role is played by the platform schedulers that have to cope with the new asymptotic scales (Sfrent and Pop, 2015). There is no universal scheduler that can be efficient on all platforms and on all environment. Moreover, collected data are complex and can be found in all forms: raw data, Web logs, streaming, environmental data, structured or unstructured data. New requirements arise for the nowadays platforms that should manage and accommodate data received at high rates. Unfortunately, large amounts of all these data are meaningless, so new ways of analysing, processing, filtering and storing are needed to cope with the new challenges (Russom, 2011).

Today's computing platforms used for data-intensive applications must manage heterogeneous resources and heterogeneous workloads. The scheduling algorithms that are used for big data have several constraints among which we can mention the deadlines of the tasks, the *makespan* of the applications, the utilization and load balancing of the resources, the data locality and so on. Therefore, there is a need for scheduling algorithms that satisfy the constraints of the applications and infrastructure of a computing platform. Furthermore, these scheduling algorithms should accommodate the large amounts of requests that need to be addressed. Finding an efficient task scheduling algorithm proves to be an NP-complete problem. As a consequence, we need heuristics that help us find a near optimal solution.

The scheduling problem that we address is stated as follows: consider a finite set of resources and an application with an unbounded number of tasks. Each task must be executed on a specific machine, having specific computational requirements and data. A machine can process one task at a time and preemption is not allowed, which means that once a task starts its execution, it can no longer be interrupted. We have an additional constraint represented by the deadlines of each task that must be taken into consideration when scheduling the tasks. We must schedule and send for execution all the tasks with minimum penalties and high throughput. In data-intensive applications, energy is consumed just to move data around without performing useful computation. Consequently, a good balance between the computing resources and performed workload is mandatory (Negru *et al.*, 2015).

Moreover, as regards the model we consider the case of data dependent tasks. But at some point, a data dependent model may enforce execution dependencies. The workload represents a large amount of complex information received at high speed, so this paper proposes a hybrid scheduler for dealing with these requirements and designed for big data environments. Related to the available resources, each machine is responsible for the execution of an infinite number of tasks; by infinite, we understand the repetitive and non-deterministic execution of tasks. In our model, we consider the case of a low machine heterogeneity and high task heterogeneity (Bessis *et al.*, 2011). A high heterogeneity degree of resources (e.g., virtual machines) has as a consequence higher energy consumption at the datacenter level (Negru *et al.*, 2016). Due to its heterogeneity regarding tasks and workloads, and data dependencies between tasks, the algorithm consist of two phases: the task selection phase and the machine selection phase for that task.

Big data processing represents the general context of the paper. For validating the proposed algorithm, we build a task scheduling simulator that supports multiple scheduling algorithms like first come first served, min-min or min-max heuristic. In this paper, we describe the integration of our proposed methodologies with real platforms, like Hadoop, OpenStack and IBM BlueMix.

To summarize our contributions in this paper, we proposed:

- a hybrid scheduling algorithm for many task computing based on min-min, min-max and earliest deadline first heuristics;
- a new measure for QoS that considers how the deadlines are respected in our proposed model;
- a set of methodologies for direct integration of the proposed model in real environments.

The paper is organized as follows. Section 2 starts with the arguments and challenges offered by big data processing models that represent a basis for our research. Then, the motivation behind our proposed model is depicted. Next, Section 3 shows the proposed model of the hybrid scheduler for many task computing. The hybrid algorithm proposed is described there. Section 4 describes the experimental evaluation and performance analysis of this new algorithm. Then, Section 5 offers guidelines of integration in real cloud environments. The paper ends with Section 6 presenting the conclusions and future approaches.

2. Background and related work

2.1. Transforming big data into smart data. In many scientific studies and use cases, the concept big data refers to large and complex datasets that cannot be accommodated by the traditional databases, processing and analysis tools in reasonable time (Zikopoulos and Eaton, 2011). For example, 90% percent of the world's data has been created in the last two years. In general, the big data term refers to scales up to petabytes and exabytes of data, and not only the datasets but also the tools that are needed to store or to process data in

real time, platforms that must ensure privacy, security and recovery of the information in the case of any imaginable situation. Challenges arise at every step: from manipulating vast volumes of complex information at high speeds, to structured, raw, semi-structured or unstructured data, to extracting the relevant information, analyzing it in real time or storing it. Currently, the data sets are continuously growing and this imposes limitations in the field of simulations or research. For enabling an efficient big data environment, not only storage and computational resources are needed, but also frameworks, like MapReduce, that can distribute the work among many nodes. Big data analytics is the method through which large data sets are examined to discover patterns, connections or meaningful information.

The *volume* of data refers to the large amounts of generated data. An outstanding increase in the amounts of data has been recorded in the last few years, from terabytes to petabytes. Vast amounts of data are generated each day. For example, Twitter generates 7 TB of data, Facebook 10 TB, CERN almost one petabyte of data every second from experiments with the Large Hadron Collider (LHC). Every type of data is stored: from structured to unstructured data. The huge amounts of data have different origins: science, telecommunications, financial, medical, environmental, surveillance, social media and social networks and so on. Sensors, science, software logs, cameras, or wireless sensor networks, mobile devices, sensors, experiments, environmental events and so on, generate these data (Manyika *et al*., 2014). These vast volumes highlight the data processing challenges: storage, analysis, processing and management of valuable content. All this is possible with the help of cloud storage, where data can be stored in different locations and processed all together by software. This property of big data plays an important part in processing phase because the scheduling algorithms should be able to adjust to these large data sets. The scheduling phase makes a big difference in getting a fast valuable content from the datasets. Even the smallest optimization to the scheduling algorithm represents a step ahead in getting the meaningful data in time.

Along with the growth in data size, the complexity and *variety* of data have increased, too. As regards data diversity, new challenges arise with the new means of generating it like sensors, smart devices, the Web, social media, experiments or applications (Zikopoulos and Eaton, 2011). Big data systems, using NoSQL databases, should handle complex data: starting with the traditional one, like relational data, and continuing with the non-traditional one, like raw data, semi-structured or unstructured data. Almost 80% of the world's data is now unstructured and traditional databases cannot be used anymore for storing and managing them (Manyika *et al*., 2014). Big data systems must be able to store, process and analyze all these various types of data: photos, videos, voice, sensor data. In the scheduling phase, the data variety plays an important role since different types of data must be processed and mapped to dedicated resources.

In terms of big data, the volume and its variety refer to other scales than before, like in the case of velocity, and so is the case of velocity, which now focuses on the speed of generating data and on how fast it can be handled. *Velocity* is defined as the speed at which the data is flowing (Zikopoulos and Eaton, 2011). Big data must provide means of storing, retrieving and processing these complex data at high speeds. The increase in data generating sources led to a constant flow of data at a speed that cannot be handled by traditional systems any more. A good example can be the social media messages that are spread in seconds. Since the time of delivery services is highly important, e.g., for the financial markets, also the speed of manipulating and analyzing complex data in real time plays a key role. As IBM people are saying, one must "perform analytics against the volume and variety of data while it is still in motion, not just after it is at rest" (Zikopoulos and Eaton, 2011). Data velocity is highly important for the scheduling part of processing big data because the scheduler must be able to adjust and deal with the high loads of data at peak times. Let us suppose that important news is released on the social networks. In several seconds, the piece of data starts to flow and so, many data of all types (text, photos, videos) are rapidly generated. Therefore, having a scheduler that can handle the peak rates of an application is crucial.

Another important aspect of big data is *veracity*. This property refers to data trustworthiness. Because the volume and complexity of data are increasing, its quality and accuracy are becoming less controllable (Normandeau, 2013). Getting the meaningful data is essential because, this way, the processing and analyzing time of relevant data is reduced. Moreover, the storage space is saved by keeping only the relevant information. The veracity characteristic plays an important role in ensuring the quality of service (QoS) of an application.

The *value* of the datasets is also important. This means that it does not matter how big the data volume is or how complex it is unless we can extract the meaningful information. Storing and processing meaningless data represents a waste of money, time, business and obtaining the relevant information becomes more difficult. Furthermore, scheduling jobs that are analyzing or processing useless data is inefficient in terms of costs (occupied resources, consumed energy) and introduces delays for getting the relevant information.

The *volatility* of big data refers to how long the data are valid and how long they should be stored. The scheduling for volatile data must be implemented for real time systems. The scheduler should take into

consideration the deadlines of the submitted jobs so that relevant data could be processed. In the case of volatile data, the datasets must be processed and analyzed in real time, otherwise one cannot obtain meaningful information from it.

2.2. Motivation behind our research. The motivation behind this paper and solving this problem of scheduling for many task computing is related to the large and complex data sets that are constantly growing. Nowadays the data are very diverse. They are generated in large amounts from all fields: environment, economics, Web logs, medicine, sensors, surveillance cameras, search indexes, social media, social networks, scientific experiments and so on. Thus, every little performance optimization makes a big difference when handling huge datasets (Negru *et al*., 2013).

In 2013, the engineers from CERN announced that in the last 20 years "CERN Data Center has recorded over 100 petabytes of physics data" (Aamodt *et al*., 2008). Furthermore, they claim from their records that the collisions in the Large Hadron Collider (LHC) have generated almost 75 petabytes of data between 2010 and 2013. The amounts of data are so large that the CERN Data Center sends them to other data centers from around the world to be processed and analyzed. Engineers must find the relevant collision among almost 15 petabytes of data generated every year. For these large data sets, data storage and fast access to data are required. The CERN data center can processes almost one petabyte of data every day. The CERN engineers say that there are 6000 changes that are performed in the database every second and more than one million jobs run daily. Furthermore, at peak rates, 10 gigabytes of data are transferred from the data center to other locations every second.

A concrete applicability for our scheduling model could be represented by the CyberWater project. The purpose of this project is using advanced computational and communications technologies to implement a new framework for managing water and land resources in a sustainable and integrative manner. The focus of this effort is on acquiring diverse data sets structured and unstructured from various sources like sensor networks, the Web, regulatory institutions, in a common digital platform that is subsequently used for storage, process and analysis to offer routine decision making in normal conditions and for helping in critical situations related to water and environment in general, such as accidental pollution in the case of flooding (Nicolae *et al*., 2014). Other examples of applications that may benefit from our scheduling model could be any of the following social network applications: Facebook, Picasa, Flick, LinkedIn. They all make part and are generators of big data volumes, high velocity complex data sets. Let us take the case of an application like Facebook, an online social networking service. A user can generate diverse data in many ways: by uploading photos, creating albums, writing comments, giving "Likes", adding friends, posting videos, sharing information. As a worldwide platform, Facebook handles huge amounts of complex information that have a high rate of appearance. Facebook stores, accesses and analyzes more than 30 petabytes of user generated data. In Tables 1–4 we can see some statistics of Facebook statistics from 2014 provided by Statistic Brain (Hepburn, 2011). From the statistics below, we can notice the high rate of updates, data diversity and large amounts of data that are handled by the Facebook platform.

2.3. Big data reduction. As the volumes and complexity of big data are increasing, it is essential to make reductions in these large amounts of data in order to get greater insights and accuracy of data for making good decisions. Moreover, it is very important to reduce the high volumes into meaningful data.

The analytics for data reduction can be divided into three categories (Delen and Demirkan, 2013):

- **Descriptive analytics**: mines the data and uses business intelligence in order to provide trending information. The descriptive analysis mines data from past or current events to provide a context for future decisions. They compute descriptive statistics (i.e., counts, sums, averages, percentages, min, max, additions and so on) that summarize certain groupings or a filtered version of the data (Reed and Dongarra, 2015). In general, they are based on standard aggregate functions in databases. The possible scenarios of using this type of analytics can be found in management reporting such as marketing, sales, finance, and so on.

- **Predictive analytics**: forecasts events based on statistical models. The predictive analysis provides various likely future scenarios based on historical and current facts for events or situations by using statistical models and data mining. Among the predictive capabilities, in the article by Waller and Fawcett (2013) the forecasting and the simulation are mentioned. This helps users to make better decisions that are based on relevant data. For predicting, the model uses trends of time-series data and correlations for identifying patterns. A good example for using this model could be the case of a company that wants to predict the customer behaviour based on customer data (Delen and Demirkan, 2013).

- **Prescriptive analytics**: makes use of optimizations and simulations to suggest actions. The prescriptive model analyzes possible actions and provides options based on the previously made descriptive

Table 1. General Facebook statistics regarding the number of Facebook users and usability.

Facebook statistics	Data
Total number of monthly active Facebook users	1,310,000,000
Total number of mobile Facebook users	680,000,000
Increase in Facebook users from 2012 to 2013	22%
Total number of minutes spent on Facebook each month	640,000,000
Percent of all Facebook users who log on in any given day	48%
Average time spent on Facebook per visit	18 minutes
Total number of Facebook pages	54,200,000

Table 2. Demographic Facebook statistics related to a user's average usability.

Facebook demographics	Data
The average number of friends per Facebook user	130
The average number of pages, groups, and events a user is connected to	80
The average number of uploaded photos per day on Facebook	205

and predictive analysis. The suggested solutions consist of a reliable path for the optimal solution together with explanations why these are the recommendations and what implications the actions might have (Waller and Fawcett, 2013). Furthermore, the prescriptive model takes into consideration the risks and offers suggestions on how to overcome them.

2.4. Many task computing (MTC). This concept was introduced by Raicu *et al*. (2008) to represent a connection between high throughput computing (HTC) and high performance computing (HPC) paradigms. In comparison with HTC, MTC uses many computing resources for a large number of short computational tasks (independent or dependent tasks). A characteristic for MTC is that the metrics considered for this paradigm are measured in seconds, e.g., FLOPS, tasks/sec, MB/s I/O rates (Raicu *et al*., 2008). In contrast, the HTC metrics are measured per month, e.g., jobs.

The problem space can be partitioned into four main categories: tightly coupled MPI (message passing interface) applications, analytics category like data mining analysis (MapReduce), the loosely coupled applications involving many tasks and the fourth category represented by data-intensive many-task computing with many tasks and large datasets. The MTC is designed to belong two main categories: big data and many tasks and the latter is represented by the many loosely coupled tasks. The typical tasks for MTC are enumerated as follows: "small or large, uniprocessor or multiprocessor, compute intensive or data-intensive" (Raicu *et al*., 2008).

The suitable applications for MTC are the loosely coupled ones that are communication-intensive but not using MPI (message passing interface) as HTC does. Raicu *et al*. (2008) claim that the HPC platforms are suitable to host MTC applications. One important fact is that the MTC supports broader categories of tasks in comparison with HTC. MTC can handle fine-grained tasks, independent or dependent tasks. In addition, tightly coupled applications and loosely coupled ones can be found together on the same MTC platform. Moreover, MTC focuses on data-intensive applications as nowadays a big difference has been noticed between the amount of processing power and the storage performance. MTC computations include multiple distinct activities, coupled via files, shared memory, or message passing.

Large MTC applications may stress the HPC hardware and software. Among the challenges that arise we can mention the "local resource manager scalability and granularity, the efficient utilization of raw hardware, shared file system contention and scalability, data management, I/O management, reliability at scale, application scalability, and understanding the limitations of HPC systems to identify suitable MTC applications." Running MTC applications on cloud systems may face challenges like internode communication performance.

There are four factors that sustain the deployment of MTC applications on petascale HPC systems: (i) the I/O subsystems of petascale systems offer unique capabilities needed by MTC applications, (ii) the cost to manage and run on petascale systems like the Blue Gene is less than that of conventional clusters or grids; (iii) large-scale systems inevitably have utilization issues, (iv) some applications are so demanding that only petascale systems have enough compute power to get results in a reasonable time-frame, or to exploit new opportunities in such applications.

3. Hybrid scheduler for many task computing in big data systems

This section introduces the model and the algorithms that were combined for obtaining our proposed hybrid

Table 3. General Facebook statistics regarding the number of Facebook users and usability.

Every 20 minutes on Facebook	Data
Links shared	1 million
Friends requested	2 million
Messages sent	3 million

Table 4. General Facebook statistics regarding the number of Facebook users and usability.

Every 60 seconds on Facebook	Data
Posted comments	510
Status updates	293,000
Photos uploaded	136,000

scheduling algorithm in big data environments. Furthermore, the pseudo-code for the proposed algorithm is presented.

3.1. General model. This section introduces the model that we are going to use in building our hybrid scheduler. For our model, we consider a heterogeneous computing environment. The workload is very diverse and so are the machine resources capabilities. We address the following problem: a finite set R of resource machines and a finite set T of tasks submitted. Let

$$T = \{T_1, T_2, \ldots, T_n\} \tag{1}$$

be the set of n tasks that are submitted. We assume that the inter-task data dependencies and preemption are not allowed. When scheduling the tasks we take into consideration also the deadlines of every task. Let

$$R = \{R_1, R_2, \ldots, R_m\} \tag{2}$$

be the set of m heterogeneous resource machines on which the tasks are scheduled. Each machine maintains a task queue with the ready tasks already submitted by the scheduler. On each machine, the tasks from the queue are scheduled in the first-come first-served order.

The set of tasks and resources are known from the beginning along with each resource capability. A resource is defined by the following parameters:

$$R_i = \{P_i, D_i, Q_i\}, \quad i = 1, \ldots, m, \tag{3}$$

where

- P_i represents the computing power of resource R_i, in MFLOPS;
- D_i represents the available disk on resource R_i, in megabytes;
- Q_i is the queue of tasks of resource R_i, that are scheduled locally in the first-come first-served order. This queue is empty at the beginning and is filled by the scheduler with tasks.

When the tasks are received for scheduling, they arrive with several requirements and parameters. Each task knows its computational and data requirements for execution. We also know the deadline time of the task. A task is defined by the following characteristics:

$$T_j = \{p_j, d_j, \mathit{arrivalTime}_j, \mathit{startTime}_j, \mathit{availableData}_j\}, \quad j = 1, \ldots, n, \tag{4}$$

where

- p_j represents the processing units needed by T_j to be executed;
- d_j represents the required disk for executing T_j;
- $\mathit{arrivalTime}_j$ is the time at which T_j arrives for scheduling;
- $\mathit{startTime}_j$ represents the latest time at which the task can be sent for execution; this represents the deadline for T_j;
- $\mathit{availableData}_j$ indicates whether or not the task has the data available so that it can start executing.

The cost function for executing a task T_j on resource R_i is stated as following:

$$\mathit{Cost}_{ij} = P_i \times p_j + D_i \times d_j, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n. \tag{5}$$

For the scheduling algorithm we need to be able to estimate the time required by a task to run on a certain machine. To this end, we need the processing (P_i) and data (D_i) capabilities of a machine and the memory (p_j) and data (d_j) requirements of a task T_j. Assuming this, the estimated time to compute for a task is:

$$\mathit{ETC}(T_j, R_i) = \frac{p_j}{P_i} + \frac{d_j}{D_j}. \tag{6}$$

Throughout this paper, we use the term "cluster" by referring to a set of heterogeneous machines with different

computational and data capabilities, interconnected with high-speed links that are able to perform different tasks with certain resource requirements.

The proposed scheduling model is a hybrid algorithm between min-min and min-max heuristics, combined with the earliest deadline first in the nonpreemptive version.

3.2. Existing scheduling heuristics. We describe various heuristics that are combined for building our hybrid scheduling algorithm. For these heuristics, we compute the estimated execution time of all the tasks on all the available resources. We consider $E_{ij} = ETC(T_j, R_i)$ the estimated execution time of task T_j, $j = 1, \ldots, n$ on resource R_i, $i = 1, \ldots, m$. We define W_i as being the previous workload on resource R_i. The time needed for R_i to finish the execution of all allocated tasks is

$$\sum_{j=1}^{n} E_{ij} + W_i, \quad i = 1, \ldots, m. \tag{7}$$

The next heuristics consider two metrics for evaluating the performance of the scheduling algorithm. These are the *makespan* and the *flowtime*. The first one represents the time when the latest task finishes. The *flowtime* is defined as the sum of all finalization times of all tasks. According to this, the formulas for computing the *makespan* and *flowtime* are (Izakian *et al*., 2009)

$$\textit{makespan} = \max_i \Big\{ \sum_j E_{ij} + W_i \Big\}, \tag{8}$$

$$\textit{flowtime} = \sum_{i=1}^{m} \sum_{j=1}^{n} E_{ij}. \tag{9}$$

The cost of the data transmission was removed from the objective function because, when we run the scheduling algorithm all tasks are ready in the system, so these transfers do not affect the scheduling and allocation processes.

Min-min is a heuristic that uses as metric the minimum completion time (MCT). This indicates that the task that can be completed earliest will be executed first. Let us define U as the set of unmapped tasks that have to be scheduled. Based on these tasks, the set of minimum completion times is computed as $C_{ij} = \min(\textit{completion\_time}(T_j, R_i))$ (Izakian *et al*., 2009). The entries of set C_{ij} represent unmapped tasks. The next step consists in selecting the task with the overall completion time among the set C_{ij}. The selected task is assigned to be executed on the corresponding resource and removed from the set of unmapped tasks. The selected resource workload is updated. This procedure is repeated until there are no unmapped tasks left. As stated in (Izakian *et al*., 2009) this heuristic minimizes the flowtime. The pseudo-code of this heuristic is shown in Algorithm 1.

The *min-max* heuristic, proposed by Izakian *et al*. (2009), uses two metrics for assigning each task: the minimum completion time and the minimum execution time. This heuristic is composed of two steps. At the beginning, the set of unmapped tasks U is considered. Firstly, the set of minimum completion times is computed for all available machines being C_{ij} (already defined). In the second phase, the task with the maximum value obtained by dividing the minimum execution time by its execution time is selected for scheduling. As demonstrated, the min-max heuristic minimizes the *makespan*. Algorithm 2 presents the pseudo-code of this heuristic.

Algorithm 1. Min-min heuristic.

```
1:  U = set of unmapped tasks;
2:  while U ≠ φ do
3:      Z ← φ;
4:      for each T_j ∈ U do
5:          for each R_i, i = 1, 2, ..., m do
6:              C_ij = W_i + E_ij;
7:          end for
8:          C_xj = min_{i=1,2,...,m} {C_ij};
9:          Z ← Z ∪ C_xj;
10:     end for
11:     Select C_qp = min_{C_xy ∈ Z} {C_xy};
12:     Allocate task T_p to resource R_q;
13:     W_q = W_q + E_qp;
14:     U ← U − T_p;
15: end while
```

Algorithm 2. Min-max heuristic.

```
1:  U = set of unmapped tasks;
2:  while U ≠ φ do
3:      Z ← φ;
4:      for each T_j ∈ U do
5:          for each R_i, i = 1, 2, ..., m do
6:              C_ij = W_i + E_ij;
7:          end for
8:          C_xj = min_{i=1,2,...,m} {C_ij};
9:          E_hj = min_{i=1,2,...,m} {E_ij};
10:         K_xj = E_xj / E_hj;
11:         Z ← Z ∪ K_xj;
12:     end for
13:     Select K_qp = max_{K_xy ∈ Z} {K_xy};
14:     Allocate task T_p to resource R_q;
15:     W_q = W_q + E_qp;
16:     U ← U − T_p;
17: end while
```

This *earliest deadline first* (EDF) scheduling algorithm considers for execution the unmapped task whose deadline is closest to the current point in time. In

our paper, we will consider the algorithm version in which task preemption is not allowed.

The *first-come first-served* (FCFS) scheduling policy is a nonpreemptive algorithm that minimizes the context switch overhead. The algorithm puts in a queue the jobs in the order of their arrival and sends them for processing in this order. This algorithm best suits the computed bound jobs and imposes high penalties to short jobs.

3.3. Proposed hybrid model. Our hybrid scheduling algorithm is a combination of the min-min heuristic, min-max heuristic and the EDF algorithms for addressing the stated problem. At each moment, there are two sets of tasks: the waiting and the ready set of tasks. A waiting task goes from the waiting set to the ready set of tasks each time the data for that task is ready.

The proposed scheduling algorithm uses the following metrics for assigning each task: the minimum completion time, the minimum execution time and the minimum deadline. At the beginning, all tasks are in a waiting list. When the data dependencies for a task are solved, it passes from the waiting list to the ready list, as shown in Algorithm 3. We begin by considering the set of ready tasks U.

Firstly, the set of minimum completion times and second minimum completion times is computed for all available machines. Between the tasks with minimum completion time and second minimum completion time, that task is selected that has the maximum execution time. In the last step, among the selected tasks in the previous step, the task with the earliest deadline first is selected for scheduling. The steps described above are repeated until the set of unmapped tasks becomes empty.

Algorithm 3. Proposed heuristic.

```
1:  WaitingTask = set of waiting tasks;
2:  ReadyTask = set of ready tasks;
3:  while WaitingTask ≠ φ do
4:    for each T_j in WaitingTask do
5:      if T_j has data ready then
6:        ReadyTask ← ReadyTask ∪ T_j;
7:        WaitingTask ← WaitingTask − T_j;
8:      end if
9:    end for
10: end while
11: U ← ReadyTask;
12: while U ≠ φ do
13:   Z ← φ
14:   for each T_j ∈ U do
15:     for each R_i, i = 1, 2, ..., m do
16:       C_ij = W_i + E_ij;
17:     end for
18:     C_xj = min_{i=1,2,...,m} {C_ij};
19:     C_yj = min_{i=1,2,...,m;x≠y} {C_ij};
20:     Z ← Z ∪ max {E_xj, E_yj};
21:   end for
22:   Select E_qp = min_{E_xy∈Z} {E_xy};
23:   Allocate task T_p to resource R_q;
24:   W_q = W_q + E_qp;
25:   U ← U − T_p;
26: end while
```

4. Experimental results

In order to be able to benchmark the new hybrid scheduler that we propose, we have designed and built a task scheduling simulator for implementing and testing the algorithm. The simulation platform supports multiple scheduling algorithms among we can mention: first come, first served, min-min heuristic, min-max heuristic, the proposed hybrid scheduler. Integrating a new scheduling algorithm to the platform is very easy.

The task scheduling simulator keeps two lists of tasks: the waiting tasks and the ready ones. The waiting tasks signify those tasks for which the data dependencies have not been solved, so they were not ready for scheduling. The ready list consists of tasks for which the data required were ready and they could be scheduled at any time from the moment that they entered the ready list. For simulating this behavior, we assumed that at every 100 milliseconds, from a range of 5 up to 5000 tasks, depending on the amount of tasks, the tasks were passing from the waiting list to the ready list of tasks. Regarding the deadlines, at every second, the deadline for all tasks in both lists, waiting and ready, were decreased by one.

The task scheduling simulator reads from an input file the configuration of the tasks and resource machine properties. This way we could simulate the task and the machine heterogeneity environment. The simulator is composed of the Task and Resource entities encapsulating the properties defined in the configuration file. Each Resource is running in a separate thread and puts in a queue the received tasks for scheduling. The tasks from the queue of each resource are executed in the first-come, first-served order. For simulating the execution, a sleep time is introduced for that thread. The sleep time equals the estimated time to compute the task on that resource. Each supported algorithm is running in a different thread until no tasks are left in the ready and waiting lists. The algorithms consider for scheduling only the tasks from the ready queue. Each scheduling algorithm puts the selected task for execution in the chosen resource's queue.

For evaluating the proposed hybrid algorithm and for comparing its performance with the other scheduling algorithms, we considered the *makespan*, the *flowtime*, the number of deadlines met and the QoS (1), as can be

noticed in Figures 1–4. We adopted

$$QoS = \begin{cases} \dfrac{make\_span}{nmd} & \text{if } nmd > 0, \\ make\_span & \text{if } nmd = 0, \end{cases} \tag{10}$$

where *nmd* is the number of missed deadlines.

In our simulation we used a setup with low heterogeneity workloads and low heterogeneity resources. The tasks execution varied from 25 milliseconds and reached up to 110 milliseconds on different scenarios.

Along with varying the number of tasks and the number of machines, we also varied the frequency at which the tasks were ready for scheduling. Moreover, we varied the deadlines of the tasks according to their number. For example, let us take the scenario with 500000 tasks in which we used deadlines of 500 seconds. Depending on the number of tasks and resources, we also adjusted the deadlines of the tasks so as data were neither too tight, nor too loose. In order to accommodate the tasks in a reasonable time, we simulated a setup with 100 machines. Moreover, the rate at which data dependencies were resolved was 10000 every 100 milliseconds. This way, we had 10000 tasks every 100 milliseconds ready for scheduling.

With respect to *makespan*, the new algorithm proposed did not produce good results compared with the other two algorithms but, in terms of *flowtime*, the algorithm's performance is close to the results of the other two.

The results prove that the proposed hybrid algorithm scheduler behaves very well in terms of meeting deadlines compared with the other two heuristics. It meets all the deadlines even for 500000 tasks, whereas the other reach up to almost 40000 missed deadlines.

As regards the measurement of the QoS, our proposed hybrid scheduling algorithm obtains the best results among the other two heuristics. This proves that for our problem, the hybrid scheduler has a good performance.

5. Integration of real cloud environments

This section describes several possible integrations of the proposed hybrid scheduler for many task computing in a couple of cloud platforms used for big data processing, such as Hadoop, OpenStack and BlueMix. Before describing the possibilities and modifications for integrating the algorithm, first we highlight the three main phases of the hybrid scheduling algorithm that we proposed: (i) for each task, select the first and second best resources that minimize the cost function (sum between the execution time and the previous workload on that machine); (ii) between the two selected resources from the first step, choose the one that minimized the execution time; (iii) select the task with the earliest deadline for execution.

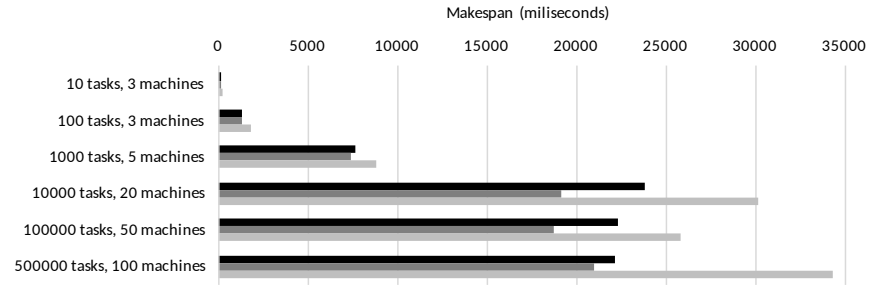
All the above steps are repeated until no tasks are left unmapped. We consider for scheduling only the tasks that have the data ready.

5.1. Hadoop integration. Hadoop is an open-source Apache software project for distributed processing of huge amounts of data across clusters of computers. It includes several modules (Shvachko *et al.*, 2010): Hadoop Common (the libraries that support the other Hadoop modules), the Hadoop Distributed File System (a distributed file system that provides high-throughput access to application data), Hadoop MapReduce (a programming model for processing and generating large data sets).

Yarn represents the MapReduce 2.0 version that decouples the resource management from the processing components. Yarn separates two major functionalities into different daemons: the *JobTracker* for resource management and the job scheduler/monitor. This way, there is a global *ResourceManager* and an *ApplicationMaster* for each application, which can be a single MapReduce job or a directed acyclic graph of jobs. Yarn brings new opportunities in the following way (Vavilapalli *et al.*, 2013):

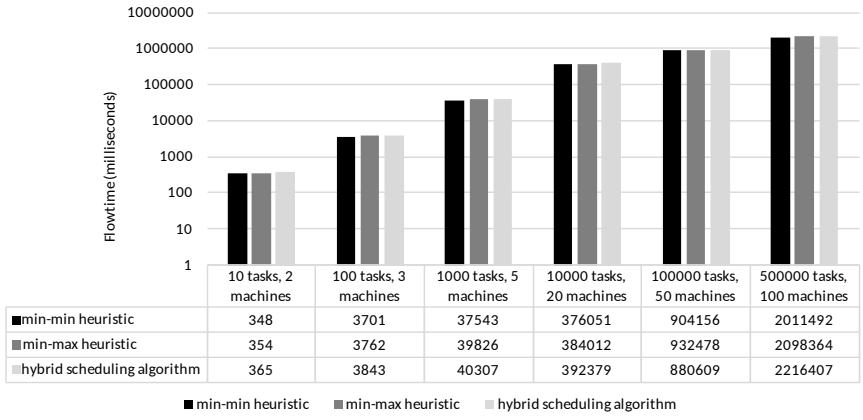
- *Scalability*: as the *ResourceManager* is responsible only for the scheduling part, it is able to manage large clusters more efficiently;
- *Compatibility with MapReduce*: The already users of MapReduce can easily deploy their application to Yarn, as no changes are required.
- *Improved cluster utilization*: There are no longer dedicated map or reduce slots so each slot can be either run as a reducer or a mapper. This new functionality drives a better cluster utilization. Furthermore, the *ResourceManager* optimizes the cluster utilization taking into consideration the capacity guarantees, fairness or SLAs.
- *Support for workload other than MapReduce*: Other programming models like graph processing or iterative modeling are supported by the platform.
- *Agility*: as the MapReduce library and the resource manager layer have been decoupled, this brings much more freedom for their evolving independently.

Regarding the data-computation framework, besides the global *ResourceManager* there are *NodeManagers* for each slave node. The *ResourceManager* is the highest level of authority for allocating the resources between applications. An *ApplicationMaster* is in charge with



	500000 tasks, 100 machines	100000 tasks, 50 machines	10000 tasks, 20 machines	1000 tasks, 5 machines	100 tasks, 3 machines	10 tasks, 3 machines
min-min heuristic	22130	22354	23856	7600	1262	98
min-max heuristic	21019	18734	19145	7420	1298	102
hybrid scheduling algorithm	34345	25820	30186	8840	1777	202

Fig. 1. Comparison results on *makespan* (milliseconds).



	10 tasks, 2 machines	100 tasks, 3 machines	1000 tasks, 5 machines	10000 tasks, 20 machines	100000 tasks, 50 machines	500000 tasks, 100 machines
min-min heuristic	348	3701	37543	376051	904156	2011492
min-max heuristic	354	3762	39826	384012	932478	2098364
hybrid scheduling algorithm	365	3843	40307	392379	880609	2216407

Fig. 2. Comparison results on *flowtime* (milliseconds).

the negotiation of the resources with the *ResourceManager*. In addition, it works with the *NodeManagers* for the execution and monitoring of the tasks.

The *ResourceManager* comprises two modules: the *Scheduler* and the *ApplicationsManager*. The former manages the allocation of resources to applications that have several requirements, e.g., capacities or task queues. The scheduler is not responsible for monitoring or tracking the applications status. Furthermore, it is not in charge with restarting the failed tasks. Based on the resource requirements of an application such as memory, CPU, disk or network, the scheduler makes the mapping between the resources and the applications. These requirements are encapsulated into a resource container entity (Sharma and Ganpati, 2015). The *ApplicationsManager* is in charge with the job submissions, the negotiation of the container for the application and restart from failure of the *ApplicationMaster*. The per-node *NodeManager* is responsible for monitoring the containers resource usage and sending feedback to the *ResourceManager*/*Scheduler*. The *ApplicationMaster* associated with each application is in charge with the negotiation for proper resource containers from the scheduler, taking their status and monitoring the progress.

Let us consider a scenario with multiple jobs submitted for execution. The jobs have different constraints (need a variable number of slots for mappers and reducers) and have different deadlines for constraints. When a job is submitted, a schedule test is performed in order to obtain the number of map and reduce slots required for its execution. The three steps described at the beginning of the section can be customized in the Hadoop system as follows:

(i) select the two jobs that have found the smallest number of available slots for execution;

(ii) between the two jobs, select the one with the shortest execution;

(iii) select the job with the earliest deadline.

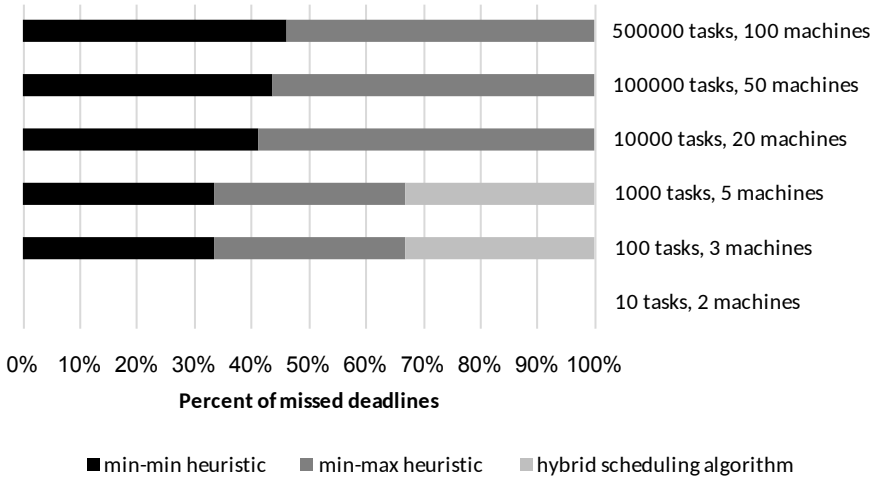


Fig. 3. Comparison of results on the number of missed deadlines.

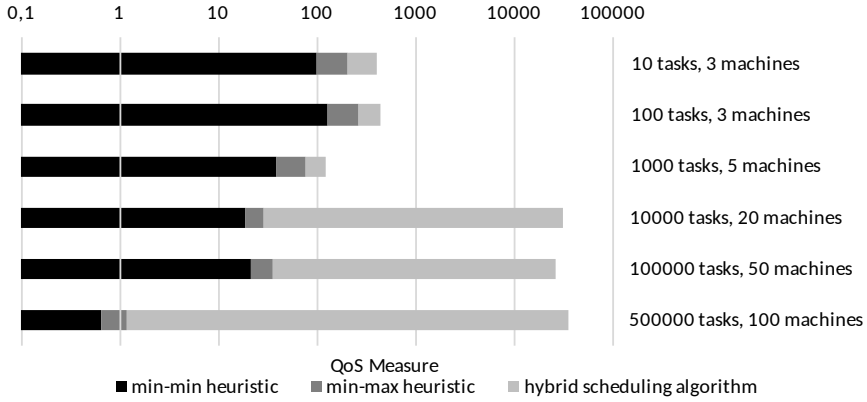


Fig. 4. Comparison of results on the QoS.

5.2. OpenStack integration. OpenStack is an open source infrastructure as a service platform for public and private clouds (Sefraoui *et al*., 2012). It is a cloud operating system that controls compute, storage and networking resources throughout a data center. We shall next focus on the computing resources provided and, in more depth, the scheduling system of the platform. Through OpenStack, enterprises and service providers can offer computing resources on demand, by provisioning and managing large networks of virtual machines. The developers that deploy cloud applications can access the resources through APIs and the administrators and users through a Web-based dashboard. The compute architecture scales horizontally on standard hardware (Litvinski and Gherbi, 2013).

OpenStack's compute project Nova includes a filter scheduler that decides on where to create a new instance based on two steps: filtering and weighting. The scheduler was designed only for the compute nodes. The scheduling flow is the following: at first, a dictionary of unfiltered hosts is constructed; after that these hosts are filtered based on some properties (image properties, compute capabilities, RAM, disk and so on) and then the host with the highest weight is chosen and put in the list of selected hosts.

Finally, it sorts the selected hosts based on the weight and provision instances on them. The filter scheduler supports many filtering strategies and gives the flexibility of building customized filtering algorithms. As they state, the weigher is a way to select the best suitable host from a group of valid hosts by giving weights to all the hosts in the list (Litvinski and Gherbi, 2013). The weights are computed per instance. For finding a prioritize order between the weights, all the weighers have to define a multiplier that is applied before computing the weight for a node. The weights are normalized before

so that the multiplier can be applied easily (Litvinski and Gherbi, 2013). The general formula for the weight is

$$\textit{weight} = w_1 \times \text{norm}(w_1) + w_2 \times \text{norm}(w_2) + \ldots + w_n \times \text{norm}(w_n). \tag{11}$$

Before customizing the proposed hybrid scheduler for the OpenStack platform, consider the following scenario. Suppose that a user wants to start more than 100 VMs at a time and each VM has different resource capabilities and different deadlines. In this case, we refer to deadline as the latest time when the VM should be started. For implementing the proposed hybrid scheduler, we have to adapt the algorithm to the architecture and workflow of OpenStack. By translating the three steps of the proposed scheduling algorithm described above into the OpenStack architecture, we should design our own filter that does the following:

- for each VM, choose the first and second hosts that best match best the resource constraints for the selected VM;
- between the selected two hosts for deploying the VM, choose the one with the least workload;
- choose the VM with the earliest deadline for scheduling.

The OpenStack VM instances scheduler requires two steps: the filtering phase and the weighing phase. For creating a new customized filter, one has to inherit the BaseHostFilter class and implement the *host_passes* method that returns true if the filter accepts the host. As parameters, the *host_passes* method receives the state of the host and the filter properties. Multiple filters can be used simultaneously. There are also some defined filters that can be extended or combined to offer the functionality required in the first step. Among them we can mention the *ComputeCapabilitiesFilter*, the *ImagePropertiesFilter*, the *RamFilter* or the *DiskFilter*.

The *ComputeCapabilitiesFilter*, which is described by Litvinski and Gherbi (2013), verifies whether the capabilities of a host match the requirements of the VM instance. The *ImagePropertiesFilter* is used to check if a host can satisfy the VM's image properties. The *RamFilter* filters the hosts based on their RAM and the *DiskFilter* based on the disk allocation, such that only the hosts that have enough disk space are considered.

5.3. Integration with BlueMix. The integration with this platform requires building a dedicated service inside BlueMix PaaS. BlueMix is the platform as a service solution provided by IBM, available in beta version since February 2014. Built on top of the IBM's Open Cloud Architecture, it offers a diverse set of services and runtime frameworks enhancing the developers to rapidly build cloud applications (Kobylinski *et al.*, 2014; Gheith *et al.*, 2016).

The service should receive as input the set of tasks and the set of resource with their capabilities, requirements and data dependencies. As a result, the service should output the order of the tasks execution and their assignment to the resources received. By implementing this service on top of BlueMix we shall have the following benefits: improve the time application or infrastructure provisioning; offer flexible capacity; address the deficiency of tech resources; reduce the total cost of ownership; enhance the exploration of new workloads such as social, mobile or big data.

6. Conclusions and future work

In this paper we achieved the goal of designing and benchmarking a hybrid scheduling algorithm for many task computing that matches our problem description. The problem that we tried to solve was scheduling tasks in a heterogeneous big data environment, taking into account the data dependencies between tasks and deadlines of the tasks. Moreover, in scheduling the phase we considered the requirements of the tasks and the capabilities of the available resources.

We described how this scheduling algorithm can be integrated with various big data platforms such as Hadoop, the OpenStack infrastructure as a service and the BlueMix platform as a service. With the help of our built-in task scheduling simulator, we compared the results of the proposed algorithm with the min-min and min-max heuristics. This way we proved that our hybrid algorithm meets the deadlines better than the other two scheduling algorithms and obtains a good QoS.

While developing the hybrid scheduling algorithm, we designed and built a task scheduling simulator in order to test the performance of the proposed algorithm. The task scheduling simulator is built in such a way that adding a new scheduling algorithm and testing it is very simple.

We can conclude that the scheduling algorithm addresses a current problem that exists in today's platforms and big data environments: the scheduling of many task computing with deadlines that have a high velocity.

A future improvement to the algorithm could be to load balance the work between the resources. For this purpose, when taking the scheduling decisions we could consider the task queue of each resource. When we refer to the task queue, we do not only think of the length of the queue, but also of the length of the tasks.

Another future enhancement to the algorithm could be to schedule chunks of some tasks at a time for the cases when there are not so many tasks in a time slice.

As a future work, the algorithm should be deployed and tested in real environments like Hadoop, OpenStack or BlueMix.

Acknowledgment

The research presented in this paper is supported by the following projects: the *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *MobiWay*: *Mobility Beyond Individualism: An Integrated Platform for Intelligent Transportation Systems of Tomorrow*, PN-II-PT-PCCA-2013-4-0321; *clueFarm*: *An Information System Based on Cloud Services Accessible Through Mobile Devices, to Increase Product Quality and Business Development Farms*, PN-II-PT-PCCA-2013-4-0870; *DataWay*: *Real-time Data Processing Platform for Smart Cities: Making Sense of Big Data*, PN-II-RU-TE-2014-4-2731.

We would also like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

References

Aamodt, K., Quintana, A.A., Achenbach, R., Acounis, S., Adler, C., Aggarwal, M., Agnese, F., Rinella, G.A., Ahammed, Z. and Ahmad, A. (2008). The Alice experiment at the CERN LHC, *Journal of Instrumentation* **3**(08): S08002.

Benziani, Y., Kacem, I., Laroche, P. and Nagih, A. (2014). Exact and heuristic methods for minimizing the total completion time in job-shops, *Studies in Informatics and Control* **23**(1): 31–40.

Bessis, N., Sotiriadis, S., Cristea, V. and Pop, F. (2011). Modelling requirements for enabling meta-scheduling in inter-clouds and inter-enterprises, *2011 3rd International Conference on Intelligent Networking and Collaborative Systems (INCoS), Fukuoka, Japan*, pp. 149–156.

Bourdena, A., Mavromoustakis, C. X., Kormentzas, G., Pallis, E., Mastorakis, G. and Yassein, M.B. (2014). A resource intensive traffic-aware scheme using energy-aware routing in cognitive radio networks, *Future Generation Computer Systems* **39**: 16–28.

Cabrera, G., Niklander, S., Cabrera, E. and Johnson, F. (2016). Solving a distribution network design problem by means of evolutionary algorithms, *Studies in Informatics and Control* **25**(1): 21–28.

Chmaj, G., Walkowiak, K., Tarnawski, M. and Kucharzak, M. (2012). Heuristic algorithms for optimization of task allocation and result distribution in peer-to-peer computing systems, *International Journal of Applied Mathematics and Computer Science* **22**(3): 733–748, DOI: 10.2478/v10006-012-0055-0.

Delen, D. and Demirkan, H. (2013). Data, information and analytics as services, *Decision Support Systems* **55**(1): 359–363.

Dimitriou, C.D., Mavromoustakis, C.X., Mastorakis, G. and Pallis, E. (2013). On the performance response of delay-bounded energy-aware bandwidth allocation scheme in wireless networks, *2013 IEEE International Conference on Communications Workshops (ICC), Budapest, Hungary*, pp. 631–636.

Esposito, C., Cotroneo, D. and Russo, S. (2013). On reliability in publish/subscribe services, *Computer Networks* **57**(5): 1318–1343.

Esposito, C., Ficco, M., Palmieri, F. and Castiglione, A. (2015). A knowledge-based platform for big data analytics based on publish/subscribe services and stream processing, *Knowledge-Based Systems* **79**: 3–17.

Esposito, C., Platania, M. and Beraldi, R. (2014). Reliable and timely event notification for publish/subscribe services over the internet, *IEEE/ACM Transactions on Networking* **22**(1): 230–243.

Gąsior, J. and Seredyński, F. (2015). Decentralized job scheduling in the cloud based on a spatially generalized *Prisoner's Dilemma* game, *International Journal of Applied Mathematics and Computer Science* **25**(4): 737–751, DOI: 10.1515/amcs-2015-0053.

Gheith, A., Rajamony, R., Bohrer, P., Agarwal, K., Kistler, M., Eagle, B.W., Hambridge, C., Carter, J. and Kaplinger, T. (2016). IBM BlueMix mobile cloud services, *IBM Journal of Research and Development* **60**(2–3): 7–1.

He, C., Li, J., Liao, Z. and Zhang, C. (2016). MPS: A multipath publish/subscribe model in information-centric network, *International Journal of Wireless and Mobile Computing* **10**(2): 130–137.

Hepburn, A. (2011). Facebook statistics, stats & facts for 2011, `www.digitalbuzzblog.com`.

Izakian, H., Abraham, A. and Snášel, V. (2009). Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments, *Neural Network World* **19**(6): 695–710.

Janiak, A., Kwiatkowski, T. and Lichtenstein, M. (2013). Scheduling problems with a common due window assignment: A survey, *International Journal of Applied Mathematics and Computer Science* **23**(1): 231–241, DOI: 10.2478/amcs-2013-0018.

Jaskóła, P., Arabas, P. and Karbowski, A. (2016). Simultaneous routing and flow rate optimization in energy-aware computer networks, *International Journal of Applied Mathematics and Computer Science* **26**(1): 231–243, DOI: 10.1515/amcs-2016-0016.

Karpowicz, M.P., Arabas, P. and Niewiadomska-Szynkiewicz, E. (2015). Energy-aware multilevel control system for a network of Linux software routers: Design and implementation, *IEEE Systems Journal* **PP**(99): 1–12.

Kobylinski, K., Bennett, J., Seto, N., Lo, G. and Tucci, F. (2014). Enterprise application development in the cloud with IBM BlueMix, *Proceedings of the 24th Annual International Conference on Computer Science and Software Engineering, Markham, Ontario, Canada*, pp. 276–279.

Kołodziej, J. and Xhafa, F. (2011). Modern approaches to modeling user requirements on resource and task allocation in hierarchical computational grids, *International Journal of Applied Mathematics and Computer Science* **21**(2): 243–257, DOI: 10.2478/v10006-011-0018-x.

Litvinski, O. and Gherbi, A. (2013). OpenStack scheduler evaluation using design of experiment approach, *16th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2013), Paderborn, Germany*, pp. 1–7.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C. and Byers, A.H. (2014). Big data: The next frontier for innovation, competition, and productivity, 2011, *McKinsey Global Institute* **5**(33): 222.

Mavromoustakis, C.X., Mastorakis, G., Bourdena, A., Pallis, E., Stratakis, D., Perakakis, E., Kopanakis, I., Papadakis, S., Zaharis, Z.D. and Skeberis, C. (2015). A social-oriented mobile cloud scheme for optimal energy conservation, *in* G. Mastorakis *et al.* (Eds.), *Resource Management of Mobile Cloud Computing Networks and Environments,* IGI Global, Hershey, PA, pp. 97–121.

Negru, C., Mocanu, M. and Cristea, V. (2015). Impact of virtual machines heterogeneity on data center power consumption in data-intensive applications, *ACM Symposium on Principles of Distributed Computing: PODC 2015, Donostia-San Sebastián, Spain*, pp. 91–102.

Negru, C., Mocanu, M., Cristea, V., Sotiriadis, S. and Bessis, N. (2016). Analysis of power consumption in heterogeneous virtual machine environments, *Soft Computing*: 1–12, DOI: 10.1007/s00500-016-2129-7.

Negru, C., Pop, F., Cristea, V., Bessisy, N. and Li, J. (2013). Energy efficient cloud storage service: Key issues and challenges, *2013 4th International Conference on Emerging Intelligent Data and Web Technologies (EIDWT), Xi'an, China,* pp. 763–766.

Nicolae, A.A., Negru, C., Pop, F., Mocanu, M. and Cristea, V. (2014). Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing, *International Conference on Network-Based Information Systems, Salerno, Italy*, pp. 221–229.

Niewiadomska-Szynkiewicz, E., Sikora, A., Arabas, P., Kamola, M., Mincer, M. and Kołodziej, J. (2014). Dynamic power management in energy-aware computer networks and data intensive computing systems, *Future Generation Computer Systems* **37**: 284–296.

Normandeau, K. (2013). Beyond volume, variety and velocity is the issue of big data veracity, *Inside Big Data*, HP Newsletter: 12 September 2013.

Raicu, I., Foster, I.T. and Zhao, Y. (2008). Many-task computing for grids and supercomputers, *Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS 2008, Austin, TX, USA*, pp. 1–11.

Reed, D.A. and Dongarra, J. (2015). Exascale computing and big data, *Communications of the ACM* **58**(7): 56–68.

Różycki, R., Waligóra, G. and Węglarz, J. (2016). Scheduling preemptable jobs on identical processors under varying availability of an additional continuous resource, *International Journal of Applied Mathematics and Computer Science* **26**(3): 693–706, DOI: 10.1515/amcs-2016-0048.

Russom, P. (2011). Big data analytics, *TOWI Best Practices Report*, Fourth Quarter.

Sefraoui, O., Aissaoui, M. and Eleuldj, M. (2012). Openstack: Toward an open-source solution for cloud computing, *International Journal of Computer Applications* **55**(3): 38–42.

Sfrent, A. and Pop, F. (2015). Asymptotic scheduling for many task computing in big data platforms, *Information Sciences* **319**(C): 71–91.

Sharma, G. and Ganpati, A. (2015). Performance evaluation of fair and capacity scheduling in Hadoop YARN, *2015 International Conference on Green Computing and Internet of Things (ICGCIoT), Greater Noida, India*, pp. 904–906.

Shvachko, K., Kuang, H., Radia, S. and Chansler, R. (2010). The hadoop distributed file system, *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA,* pp. 1–10.

Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S. *et al.* (2013). Apache Hadoop: Yet another resource negotiator, *Proceedings of the 4th Annual Symposium on Cloud Computing, Santa Clara, CA, USA*, p. 5.

Waller, M.A. and Fawcett, S.E. (2013). Data science, predictive analytics, and big data: A revolution that will transform supply chain design and management, *Journal of Business Logistics* **34**(2): 77–84.

Zikopoulos, P. and Eaton, C. (2011). *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1st Edn., McGraw-Hill, New York, NY.

Laura Vasiliu received her MSc in 2014 in the field of parallel and distributed algorithms at the University Politehnica of Bucharest, Faculty of Automatic Control and Computers. Her research interests are in resource management in distributed systems, as well as design and implementation of cloud applications.

Florin Pop is a professor at the Department of Computer Science and Engineering at the University Politehnica of Bucharest. His general research interests are large-scale distributed systems (design and performance), grid computing and cloud computing, peer-to-peer systems, big data management, data aggregation, information retrieval and ranking techniques, and bio-inspired optimization. He is also a scientific researcher within the National Institute for Research and Development in Informatics (ICI), Bucharest, Romania.

Catalin Negru is a PhD student at computer science and a systems engineer at the Computer Science and Engineering Department of the University Politehnica of Bucharest. His research interests include cloud computing, data storage, energy efficiency, resource management and cost optimization. His PhD thesis is focused on cost optimization in cloud storage systems through resource management methods and techniques.

Mariana Mocanu is a professor of computer science at the University Politehnica of Bucharest. She coordinates the team for interoperable products and services for decision support, based on geospatial data, and has a long experience in developing information systems for industrial and economic processes, and in project management. She does teaching for both undergraduate and master's degrees in software engineering, systems integration, software services and logic design. She is the coordinator of the H2020 project: *Data4Water—Excellence in Smart Data and Services for Supporting Water Management*.

Valentin Cristea is a professor in the Computer Science Department of the University Politehnica of Bucharest. His main fields of expertise are large scale distributed systems and e-services. He teaches courses, supervises PhD students, leads projects, and is active in the research related to these topics. He is a member of the IEEE and the ACM.

Joanna Kolodziej, PhD, DSc in computer science, graduated in mathematics from Jagiellonian University in Cracow (Poland) in 1992, where she also obtained her PhD in theoretical computer science in 2004. The main topics of her research are evolutionary computations, mathematical modeling of stochastic processes, grid and cloud computing, intelligent networking, scalable computation, multi-agent systems, and global optimization meta-heuristics.

Received: 26 November 2016
Revised: 20 February 2017
Re-revised: 14 March 2017
Accepted: 20 March 2017